卒　業　研　究　報　告　書

平成25年度

研究題目

ソフトウェアレビューにおける
読み方の教示によるレビュー効率の変化

指導教員　　上　野　秀　剛　　助教

氏　　名　　應　治　沙　織

平成26年1月30日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

ソフトウェアレビューにおける
読み方の教示によるレビュー効率の変化

上野研究室　應治沙織

本研究の目的は，ソフトウェアレビューの1つであるソースコードレビューの効率を向上させる要因として，レビュー時の視線の動きを用いることが有用であるかを明らかにすることである．本研究ではレビュー対象を精読する際の視線の動きを計測するため，視線計測装置を含むハードウェアコンポーネントとソフトウェアツールCrescentから成る実験環境を用いる．この環境ではレビュー作業者がソースコードのどこを見ているかを時系列に記録し，再生できる．実験は予備実験と本実験に分けて行う．予備実験で計測したレビュー能力を元に被験者を2つのグループに分割する．そして本実験では片方のグループにのみ，レビュー開始時の読み方を教示してからコードレビューを行い，レビュー効率の差を計測する．実験の結果，教示を行ったグループは教示を行わなかったグループに比べてバグ発見効率が向上した．このことから，読み方の教示はコードレビュー効率を向上させる要因となるといえる．

目 次

1 はじめに

ソフトウェアレビューとは，プログラムの中に混入している誤り（バグ）を検出するために，開発者がソースコードや設計書を精読することである．通常，ソフトウェアレビューは人間のレビュアーがプログラムのコンパイルや実行をせず，画面上，あるいは紙上のソースコードに対して行う．ソフトウェア開発において結合テストやシステムテストのような後工程で誤りが見つかった場合，その除去には多くの労力とコストがかかるため，開発初期に行うことのできるコードレビューは開発コストの削減という面において重要である．

従来の研究では，過去のレビュー経験に基づくチェックリストを用いるChecklist-Based Reading(CBR) 手法 [1] やいくつかの異なった立場，視点からコードレビューを行うPerspective-Based Reading(PBR) 手法 [2]，ユーザの視点からレビューを行うUsage-Based Reading(UBR)[3] などの手法が提案されている．これらの手法はいずれも作業者にある特定の基準を定め，ドキュメントを読むように定義されており，今までに性能を比較するため多くの実験が行われている．

従来の研究においてレビュー効率の評価実験結果に幅が見られるのは，人的要因がレビューに与える影響が大きく，また個人の性能差がレビュー効率に与える影響がレビュー手法の与える影響よりも大きいためだと考えられる．

そこで，本研究ではソフトウェアレビューの中でもソースコードを対象とするソースコードレビュー（以後，コードレビューとする）を対象に，レビュー効率を向上させることを目的とする．本研究ではコードレビュー時の作業者の視線の動きに注目する．通常ソフトウェアレビューは目視で行われるのでソフトウェアレビューにおける視線移動は作業効率に大きく影響を与えるといえる．本研究では作業者の視線の動きに着目し，作業者に作業開始時に読み方を教示することがコードレビュー効率の向上に繋がるか被験者実験により調査する．この研究からソフトウェアレビュー効率を向上させるレビュー対象物の読み方が分かれば，将来的にはソフトウェア開発における人件費の削減につながり，より安価にソフトウェアを提供できるようになるといえる．

本論文の構成は以下のとおりである．2章では関連研究について述べる．3章ではソフトウェアレビューにおける視線移動の教示について述べる．4章では実験環境や実験手順などを示し，5章では実験結果を述べ，考察を行う．6章ではまとめと今後の課題について述べる．

2 関連研究

2.1 レビュー効率を測定する研究

レビュー効率を上げるための手法として従来の研究では複数の手法が提唱されている[1, 2, 3]. 例として過去のレビュー経験に基づきチェックリストを作成するChecklist-Based Reading (CBR)[1]や，作業者が対象ソフトウェアに関わるいくつかの異なった視点（ユーザ視点，プログラマ視点，設計者視点など）からレビューを行うPerspective-Based Reading(PBR)[1]，PBRの一種であり，ソフトウェアのユーザの視点からレビューを行うUsage-Based Reading(UBR)[3]などがある．松川らはCBRとPBRの性能の比較を実験で評価した[4]. 実験では，様々な種類の図を用いてソフトウェアの体系を視覚的に表現するソフトウェア設計開発言語であるUMLを用いた設計仕様書に対し，CBRとPBRの性能を比較した．実験の結果を図1に示す. グラフはPBRが図と図の関連性や一貫性に関する意味的なバグを発見しやすく，CBRは構文的なバグが発見しやすいことを示している．

他の同様の研究ではUBRとCBRを比較し，UBRのほうが優れている[5]とされる一方，これらの手法には効率に明確な差はないともされている[6]ため，これらの手法についてはレビュー効率の違いについて明確な結果が得られていない．

従来の研究においてレビュー効率の差が明確に測られていない理由は，コードレビューが人的要因に大きく影響を受けるためと考えられる．例えば文献[7]では，UBRとCBRの誤り発見率（発見した誤りの数／誤りの総数）を比較し，図2のように様々な種類の誤りにおいてUBRのほうがCBRよりも優れていることを示している．

一方で，それぞれのグラフにおいて点線で示されているように，同じ手法を用いている際の個人差は手法の差よりも大きく広がっている．このような個人ごとの性能の差については従来よく研究されていない．そのため作業者個々の能力向上に繋がるコードビュー方法を発見することは重要な課題である．

2.2 視線計測器を用いた能力評価

視線計測による分析は主に初心者と熟練者の違いを分析することを目的に，特に認知科学の分野においてよく用いられる[8, 9]. 村田らは自動車運転時に発生する危険予知における熟練者と初心者の視線の動きを比較・分析している[8]. 実験では，自動車運転状態の静止画を運転熟練者と初心者に見せることで危険予知を行わせ，その状態の視線運動を解析している．実験の結果，カーブにおける熟練者と初心者の視線移動の違いが顕著になり，危険度がカーブよりも相対的に低く，運転しやすいと考えられる直線道路や交通量の少ない道路では熟練者と初心者の視線移動の違いが少なくなった．岩城らはデッサンの描画時における熟練者と初心者の視線移動の違いを分析している[9]. 視線計測装置を用いた実験で，

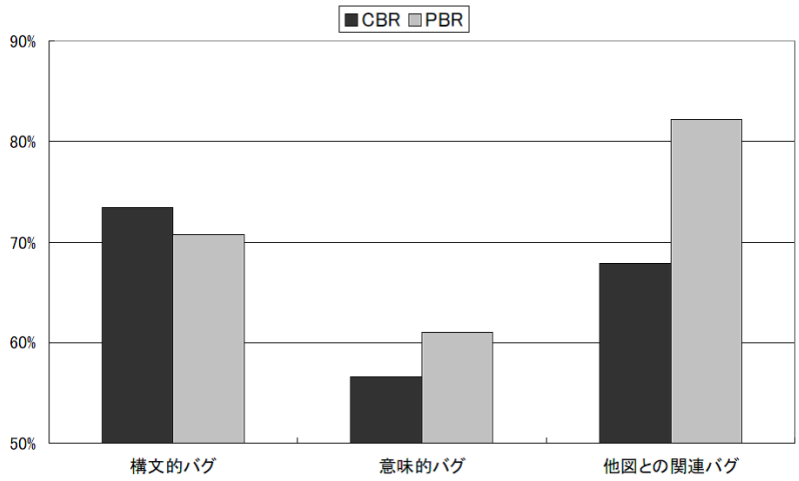

図 1 CBR と PBR のバグ検出率 [4]

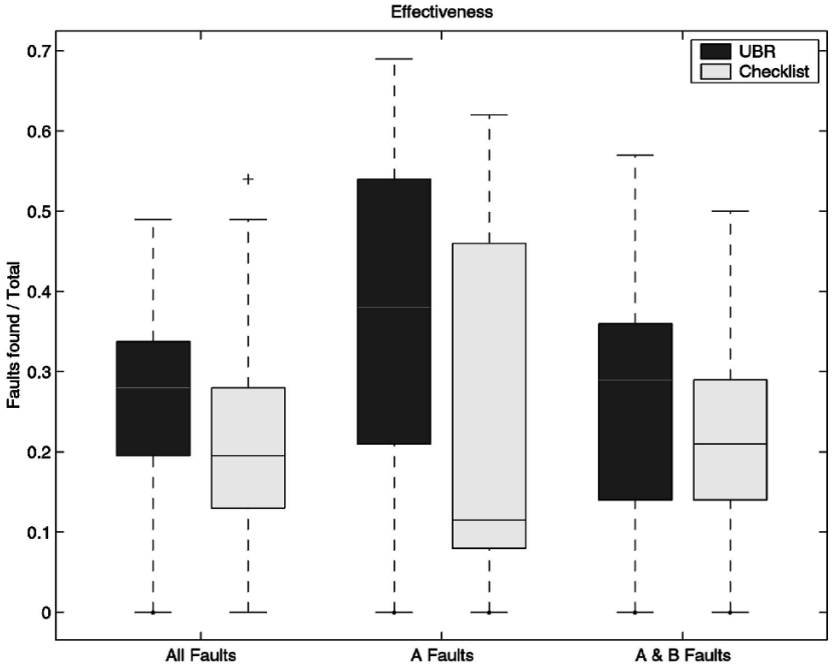

図 2 UBR と CBR の誤り発見率 [7]

デッサン熟練者と初心者のデッサン中とモチーフを見る際の視線移動を分析している．実験の結果から，熟練者はデッサンにおいて配置を決定するとモチーフをほとんど見ずに描き，それに対し初心者はモチーフを見ている時間が長くなるという違いが現れた．これは初心者は熟練者に比べモデルが頭に入っていないため，モチーフを見る時間が長くなるものと考えられた．

ソフトウェア分野においても，計測対象として視線移動を用いた研究が行われている [10, 11, 12, 13, 14]．Zhai らはコンピュータ入力において視線を用いる手法を

提案しており，マウスなどの従来の計測対象に比べ被験者にかかる疲労を軽減させることができると示している[10]．中道らはWebページ閲覧時のユーザビリティ評価に視線情報を利用している[11]．Webページのユーザビリティの評価指標に実験時に記録した視線データを用いることで，Webページのユーザビリティについてより多くのコメントが得られることを明らかにした．Steinらはソフトウェアのデバッグ作業者に他の作業者の視線移動を見せることが作業の助けになるということを示している[12]．分析結果から，他の作業者の視線移動を見てからデバッグを開始した作業者は視線移動を見なかった作業者よりも早くバグを発見できるということを明らかにした．

また，上野の研究ではコードレビューにおいて作業者がレビュー中にソースコードのどこに注目しているかを測るため，視線情報を用いている[13]．実験では被験者に仕様書，設計書，ソースコード，コードチェックリストを提示し，被験者の作業中の視線を測定した．実験の結果，レビュー開始時にコード全体を上から下に向かって眺める動作を行わない作業者は誤り検出までの時間が長くなる傾向が発見された．吉本の研究ではコードレビュー時に作業者に効率の良い視線の動かし方を教示し，教示した作業者と教示しなかった作業者で作業効率に変化が生じたかを検証している[14]．結果，教示をした作業者は誤りを発見する効率は上がったが，正しく誤りを発見できたかという点においては明確な差は得られなかった．

本研究はコードレビューにおいて作業者の視線の動きを測定するという点において上野や吉本の研究と類似しているが，本研究では実験で用いるレビュー対象のプログラムに100行程度の長い行数のものを用い，言語をJava言語にしている．比較的長いソースコードを用いることでより実際のソフトウェア開発現場と作業環境を類似させた実験を行う．また，上野や吉本の研究では手続き型言語であるC言語のプログラムを実験に用いていたが，本研究ではオブジェクト指向言語の一種であるJava言語を用いて実験する．さらに，コードレビュー時に作業者に教示する視線の動かし方についても従来より明確な指示を出した．

3 ソフトウェアレビューにおける読み方の教示

本研究ではソフトウェアレビューにおけるレビュー対象物の読み方を作業者に指示し，レビュー効率が向上するか調査する．本研究ではレビュー開始直後の読み方に着目し，プログラムの仕様とソースコードの全体像を把握するよう全体を見てからレビューするよう作業者に教示する．プログラムの全体像を把握することで効率的に誤りを発見し，作業効率が向上すると考えた．

吉本の研究[14]では，作業者にプログラムの全体像を把握してもらうために，実験開始時に「ソースコードをざっと見る」ように指示している．実験の結果，ソースコードを注視することがバグ発見にかける時間を短縮させることが分かった．表1に吉本が行った実験の結果を示す. 表1は読み方の教示をしたグループと教示をしなかったグループのソースコードごとのバグ検出時間を表す. この表からソースコード4以外のレビュー時間が，教示をしたグループの方が短くなっていることがわかる．以上の結果から，読み方の教示はコードレビュー効率を向上させるために有用であるといえる．

従来研究では教示内容が曖昧であり，実験設定において設計書を用いず，ソースコード自体も短く，混入したバグも1プログラムに対し1つと少なく，検出できなかった作業者も5分で実験を終了していたために正確な計測・比較ができなかったと考えられるので，本研究ではそれらの設定を変更して実験を行う．

また，プログラムの内容を理解するには設計書の内容を把握し，プログラムに対する要求を理解した上でソースコードを読む必要がある．そのため本研究では作業者にレビュー開始直後に設計書を精読した後にソースコード全体を見るように指示した．加えて設計書に書かれたメソッドの設計とソースコードに書かれたメソッドの動作が等しいかを確認するため，設計書とソースコードを比較し，設計通り実装されているか確認するように指示した．

なお，コードレビューにはレビュー中に行う指摘フェーズの他に，レビュー前に提示物やチェックリストに目を通す事前準備フェーズを行うことがある．今回の実験では教示内容を行っている間も誤り発見は可能であるため事前準備フェーズであるとは言えない．しかしチェックリストを提示したり，設計書をよく読むなど事前にする作業を教示している点から厳密に教示内容が指摘フェーズに含まれているともいえないので，今回行った読み方の教示は指摘フェーズに事前準備フェーズが含まれた教示であると考えられる．

表1 ソースコードごとのバグ検出時間 ([14])

	コード1	コード2	コード3	コード4	コード5
教示無し[秒]	178.3	247.8	167.3	142.0	214.8
教示有り[秒]	105.3	110.5	128.5	152.8	162.5

4 実験

実験は予備実験と本実験の2つに分けて行う．予備実験では被験者のコードレビュー能力を測るため，2つのJavaのソースコードを用いてコードレビューを行ってもらう．本実験では，視線の教示を行うグループと行わないグループの2つに被験者を分ける．2つのグループ間でコードレビューの平均能力が等しくなるように予備実験の結果を参考にグループを分割する．

予備実験，本実験共に使用する実験環境について次節で説明する．

4.1 実験環境

本研究では，先行研究が構築した視線計測環境を用いてレビュー中の視線移動を計測する．視線計測構築は，非接触アイマークレコーダEMR-AT Basic+を含むハードウェアコンポーネントと，ハードウェアを制御するソフトウェア，コードレビュー実験用のソフトウェアCrescent (Code Review Evaluation System by Capturing Eye movemeNT) から構成される．

実験環境の構成を図3に示す．本研究で用いる実験環境ではEye cameraで撮影した作業者の眼球映像から，視線計測装置を通して視線座標を取得する．Crescentは視線計測装置が計測した視線座標から注目している提示物の行番号と注視時間を割り出し，記録する．

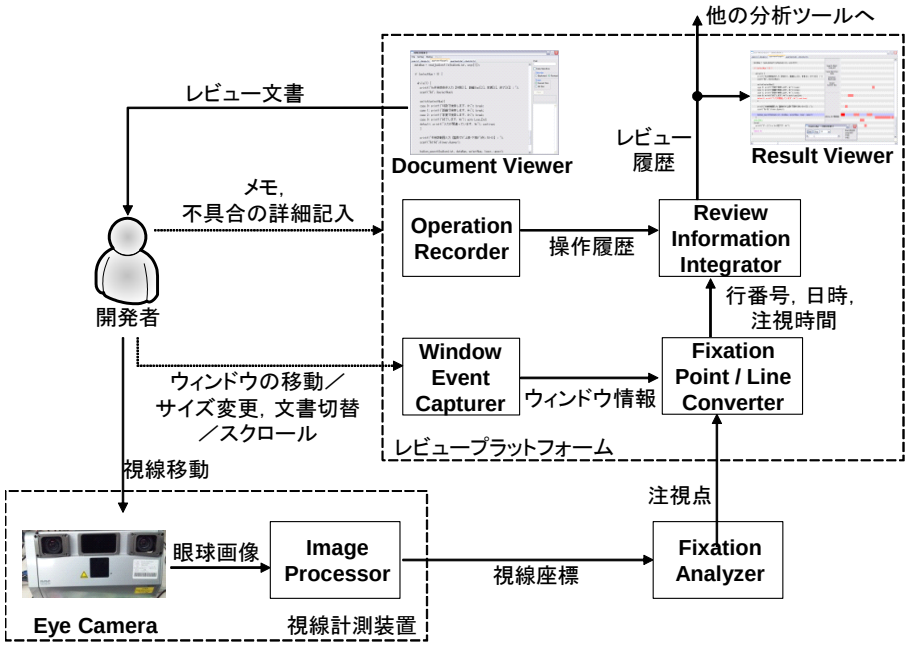

図3 視線計測器Crescentを含む実験環境の構成

4.1.1 ハードウェアコンポーネント

ハードウェアコンポーネントは視線計測装置と作業者に提示するプログラムを表示する提示用パソコン，作業者の視線の動きを計測する作業用パソコンからなる．

本研究ではソースコードを精読する作業者の視線の動きを計測するため，作業者の視線の動きを行単位で精密に識別する必要がある．そこで本研究では高解像度で精度の高い計測が可能であるnac Image Technology社[1]の非接触アイマークレコーダEMR-AT Basic+を使用した．

EMR-AT Basic+の動作画面を図4に示す.

図4の画面では被験者が作業中に提示物のどこに注目しているかを計測している．接触型の視線計測器とは違い，本実験で用いる非接触型の視線計測器では作業者のコードレビューを妨げることがなく，作業者にかかる負担を軽減することができる．EMR-AT Basic+は検出分解能0.3°，検出レート60Hzで計測を行う．ソースコードを表示するために液晶ディスプレイ (EIZO FlexscanS1721) を解像度1024×768で使用した．計測誤差は当環境においてディスプレイ上で約5.4pixelとなる．これはフォントサイズ12ptでコードを表示したとき約0.45行に相当する．

また，頭部を固定できる椅子を採用することで作業中に被験者の頭が動くことで生じる計測誤差を最小にした．

使用したパソコンは作業者が使用する作業用のパソコン2台と計測者が被験者の視線データを計測するパソコン2台の計4台である．EMR-AT Basic+も2台あるため，実験では2名の被験者を同時に計測することができる．作業用のパソコンはWindowsXP Intel(R) Core(TM)2 CPU 6400 @2.13GHzのパソコンが2台，データ計測用のパソコンはWindowsXP Intel(R) Core(TM)2 Duo CPU E4600 @2.40GHz，Windows7 Intel(R) Core(TM)2 Duo CPU　E7400 @2.8GHzが1台ずつである．

4.1.2 ソフトウェアコンポーネント

ソフトウェアコンポーネントはEMR-AT Basic+制御ソフトとCrescentからなる．

制御ソフトはEMR-AT Basic+から視線移動データを受信し，ファイルに保存する．視線移動データはEye cameraが撮影した眼球映像から割り出されたディスプレイの絶対座標で表された注視点をソースコードの論理行番号に変換したデータと注目時間である．

また，制御ソフトは注視点データを解析し，停留点を得ることができる．停留点とは一定時間継続して注視点が留まった一定の範囲のことを指し，一般に計測対象者はその位置を意識的に見ていると解釈される．当環境では直径30pixelの円内に50ms以上の注視点が留まった場合に，その中心を停留点とみなす. これによ

[1] http://www.nacinc.jp/

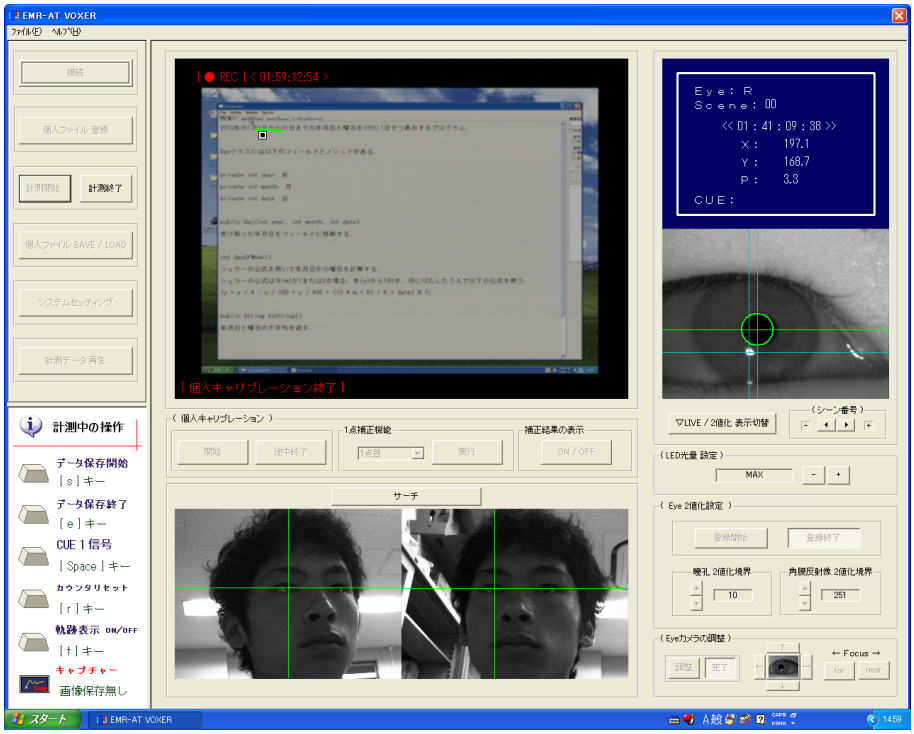

図 4 EMR-AT Basic+

り，計測対象者が認識なく対象上を通過したのか，対象を意識して注目したのかを区別することができる．

Crescent は視線計測器が測定した視線データを行単位のデータに変換するツールである．Crescent の作業画面を図 5 に示す．図 5 のウィンドウはレビュー対象となるソースコードやレビュー時に利用する他の文書がタブで分けて表示される．

また，レビュー中に誤りを発見した対象者が誤りのある行をダブルクリックすると，誤りの報告用テキストボックスが表示され，そこに誤りの内容を記述できる．

計測対象者はウィンドウに表示されたソースコードをレビューし，その際の視線移動を計測システムが計測する．Crescent はレビュー開始と終了時にアイマークレコーダにリセット信号を送信することで，計測を開始・終了する．

また，Crescent はレビュー結果を図 6 のようにグラフ形式で表示することもできる．

図 6 で示しているように，結果を表示するウィンドウではレビューしたソースコードとバーチャートで示された被験者の視線移動を表示する．また，時系列情報を使用し，バーをハイライトすることで視線移動の再生が可能である．任意時間からの再生やスロー再生をすることも可能である．くわえて，時系列情報や停留点データ，停留行データを CSV ファイルとして出力することも可能である．

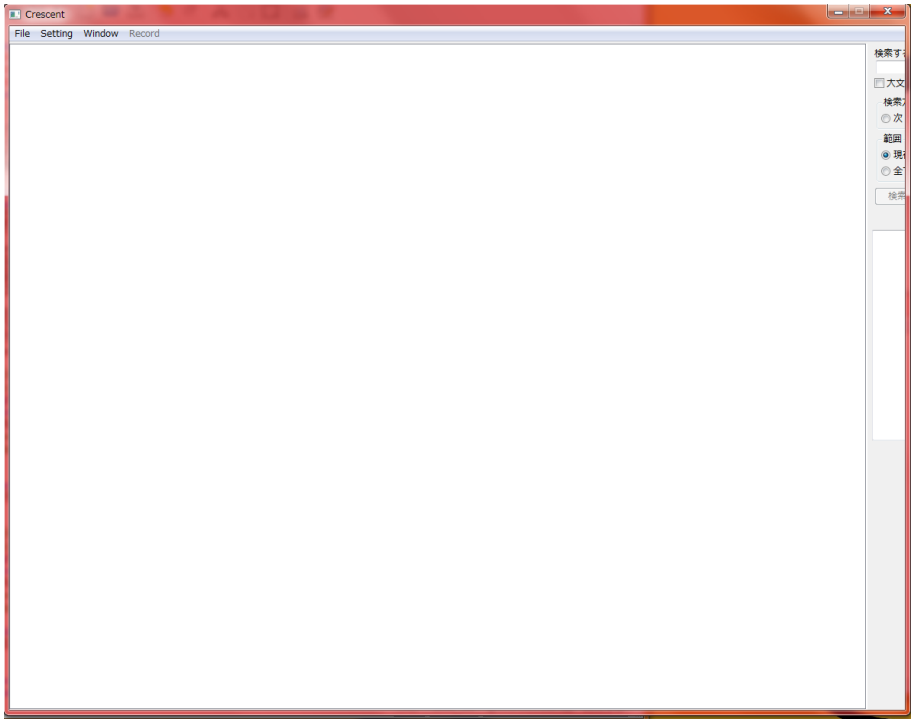

図 5 作業画面

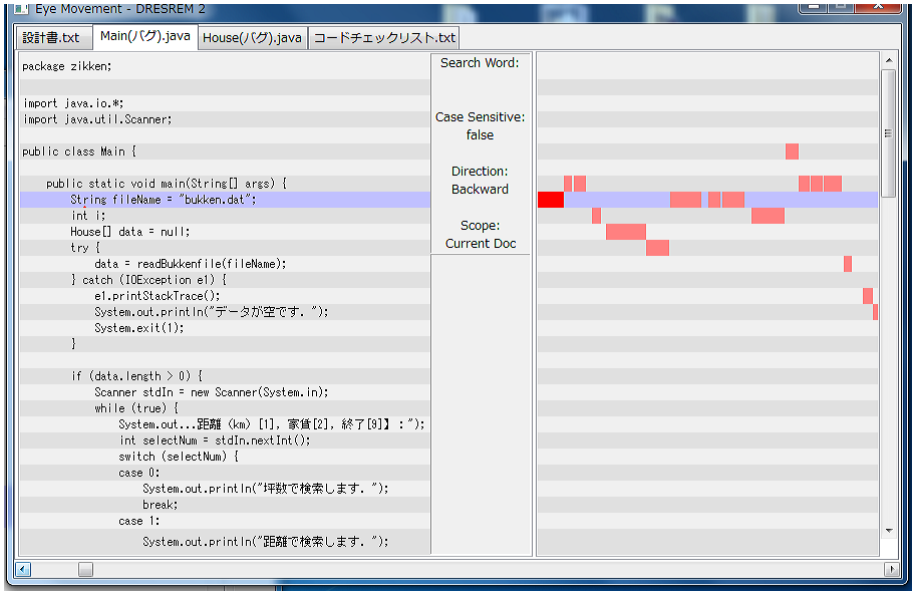

図 6 実験結果の解析

4.2 実験手順

実験手順を図7に示す. 実験は予備実験と本実験に分けて行った.

予備実験では被験者のレビュー能力を測るためレビュー開始時の読み方の教示をせずにコードレビューを行う.

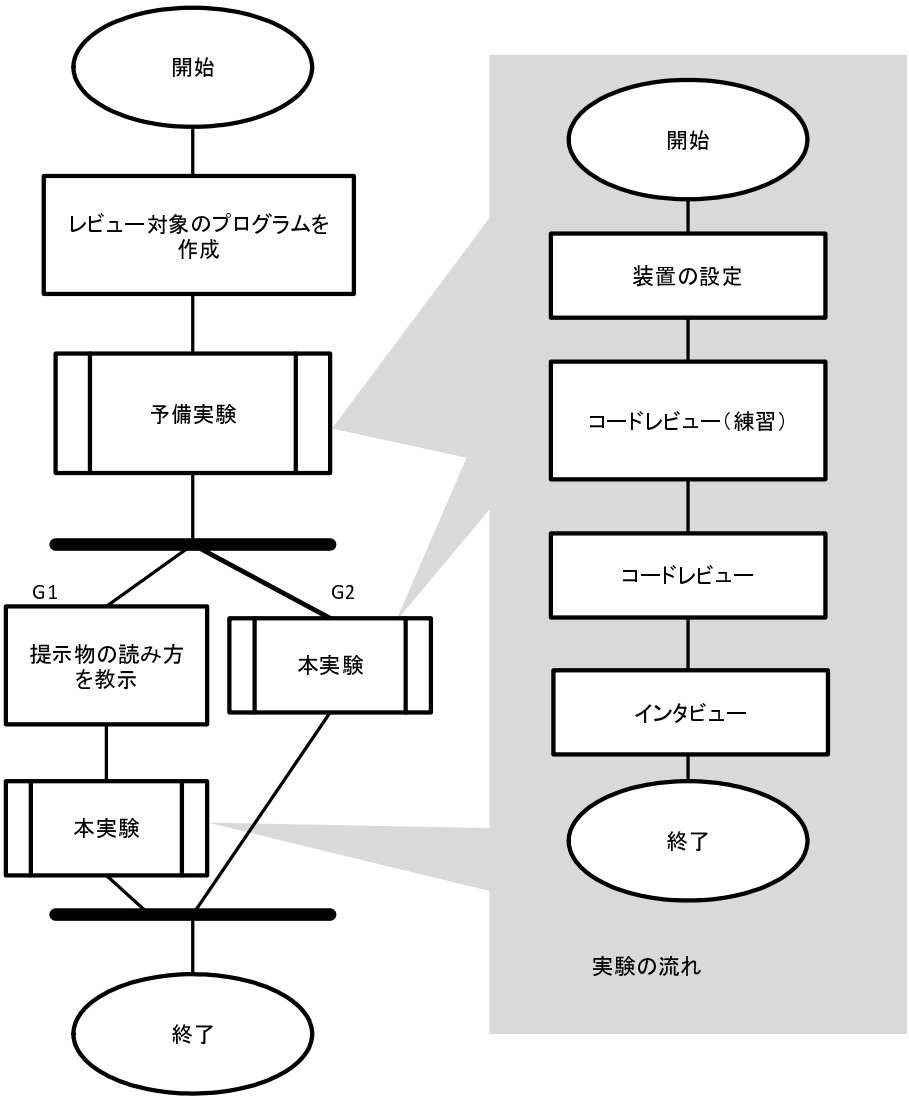

図 7 実験手順

次に，予備実験で測定した被験者の誤り発見効率をもとに，レビュー能力がほぼ等しくなるように被験者全体を2つのグループ”教示有り”と”教示無し”に振り分ける．本実験では”教示有り”グループにのみレビュー開始時の読み方を教示し，教示しないグループとの作業効率を比較した．本実験で”教示有り”グループに対して行った読み方の教示を以下に示す.

1. 設計書をよく読んでください
2. コード全体にざっと目を通してください
3. 設計書に書かれているフィールド，メソッドが正しくソースコードに実装されているか確認してください

3章でも述べたとおり，昨年の研究では教示の内容が曖昧で被験者に教示の意

図が伝わらなかったと考え，本研究ではプログラム全体を見渡す読み方を「プログラムの構造を把握するための動作」であると捉え，その読み方を十分に理解してもらうためより細かい教示内容とした．

予備実験ではレビュー時間は制限せず，被験者に作業を終了する旨を自己申告してもらう．作業の終了は作業者がコードレビューを終え，誤りをすべて発見し終えた，またはこれ以上レビューしても誤りを見つけることができない，と判断した時に行う．

予備実験では作業者が実験環境に慣れるという目的もあったため，時間制限を設けない．また，被験者を実験環境に慣れさせるため，練習用のプログラム1つのコードレビューも行わせる．

本実験では制限時間内でのレビュー効率を分析するため，時間制限を設け，40分間でコードレビューを行い，40分に達しない時点で実験を終了することは認めない．これは，時間を無制限に設けると何時間もかけてバグを発見した作業者と短時間で発見した作業者を同じものとして扱うことになり，レビュー効率を正確に測るには不適切と考えたからである．そこで，今回の実験では制限時間を設け，時間内のレビュー効率を比較・検証する．

また，被験者はJavaのプログラミングについての基本的知識を持ち合わせていると考えられる，プログラミング経験が1年以上ある学生11人を対象とする．

4.3 予備実験

4.3.1 レビュー対象

予備実験では2つのJavaプログラムのレビューを行う．予備実験で使用したプログラムを表2に示す．また，使用したJavaプログラムのソースコードや設計書，プログラム内に含めたバグは付録に記す．

4.3.2 分析に用いる指標

予備実験の目的は，本実験でグループ全体としてのレビュー効率がほぼ等しくなるよう被験者をグループ分けするために，被験者のコードレビューの力量を測ることである．予備実験の結果からレビュー効率が等しくなるよう被験者を2つのグループに分類することで，本実験で比較実験を行う際に，読み方の教示による作業効率の違いを明確に割り出すことができる．

グループ分けの基準として，被験者が発見したバグ数とプログラムに含まれるバグ数の比率（バグ発見率）を用いる．バグ発見率の計算式を以下に示す．

$$\text{バグ発見率} = \text{作業者が発見したバグ数} / \text{プログラムに含まれるバグの総数} \quad (4.3.1)$$

表 2 予備実験で使用したプログラム

実験	仕様	提示物	行数	バグ数
予備実験	2013 年の 7 月 1 日から 31 日までの年月日と曜日を 1 行に 1 日ずつ表示する	Day.java	31	3
		July.java	16	2
		設計書	25	0
		チェックリスト	37	0
予備実験	貯金の残高を表示する	Account.java	37	5
		Bank.java	41	1
		設計書	37	0
		チェックリスト	37	0
練習	1 から n までの整数の和を表示する	Main.java	18	0
		Sum.java	14	1
		設計書	12	0
		チェックリスト	37	0

バグ発見率はグループ分けに用いるほか，予備実験と本実験でのレビュー効率がどれだけ向上したかの判断にも用いる．

4.4 本実験

4.4.1 レビュー対象

本実験も予備実験と同様に Java プログラム 1 つのレビューを行う．

本実験で使用するレビュー対象プログラムは実験結果を正確に比較するため，グループ“教示有り”，“教示無し”で同一のものを使用した．また，今回の研究ではより実際のソフトウェア開発現場で研究結果を反映，役立てることができるよう，プログラムに含まれるソースコードの行数が合計して 100 行程度となる比較的長いプログラムを使用して実験を行う．

本実験で使用したプログラムを表 3 に示す．

表 3 本実験で使用したプログラム

実験	仕様	提示物	行数	バグ数
本実験	物件を表示する	Main.java	104	6
		House.java	34	1
		bukken.dat	20	0
		設計書	30	0
		チェックリスト	37	0
練習	n 段のピラミッドを表示する	Pyramid.java	14	1
		Main.java	11	0
		設計書	14	0
		コードチェックリスト	37	0

4.4.2 分析に用いる指標

実験結果の分析ではグループ別のバグ発見率を用いる.

また, バグ発見率に加え, Welchの検定による分析も行った. Welchの検定とは有意水準を用いて独立する2標本の平均値の差を比較する検定である. 有意水準とは帰無仮説が正しいときにそれを誤って棄却する確率のことを指す. 有意水準は通常0.05, 0.01などで設定され, この場合tを境界値とし, 両側確率pで考える. 有意水準が設定数以下であった場合, 2つのデータ間に有意差があると判断することができる.

5 結果と考察

5.1 予備実験結果

予備実験における各被験者のバグ発見率を表4に示す.

表4をもとに被験者を本実験のグループ”教示有り”と”教示無し”に分割した. 結果, ”教示有り”には被験者a,b,e,f,g,i（平均バグ発見率72.6%）が, ”教示無し”には被験者c,d,h,j,k（平均バグ発見率72.7%）が割り振られた.

表4予備実験の結果

被験者	バグ発見率	グループ
被験者a	72.7%	教示有り
被験者b	63.6%	教示有り
被験者c	90.9%	教示無し
被験者d	54.5%	教示無し
被験者e	63.6%	教示有り
被験者f	90.9%	教示有り
被験者g	72.7%	教示有り
被験者h	63.6%	教示無し
被験者i	72.7%	教示有り
被験者j	63.6%	教示無し
被験者k	90.9%	教示無し
平均	72.7%	
標準偏差	12.9	

5.2 本実験結果

本実験の結果である教示有りのバグ発見率を表5に, 教示無しのバグ発見率を表6示す. 表5, 表6より, 教示有りグループのバグ発見率が平均で66.7%, 教示無しグループが60.0%と, 6.7%の差が見られた. これは, 読み方の教示をしたことでプログラムに対する理解度が上昇したため生じた差であると考えられる.

2つのグループに対してWelchの検定を行った結果, $p = 0.43$で有意な差は見られなかった. これは被験者数が少なかったためと考えられるが, 平均して6.7%と大幅な変化がみられるため, 被験者数を増やして実験をすれば有意な結果が得られると考えられる.

次に, 各グループの予備実験と本実験におけるバグ発見率の差を比較する.

予備実験, 本実験の難易度差によって不正確な比較検証を行わないために, 予備実験における全被験者の平均バグ発見率で本実験のバグ発見率を正規化する. まず, 被験者数をn, i番目の被験者$(i = 1, 2, \ldots, n)$の予備実験におけるバグ発見率をS_i, 本実験におけるバグ発見率をM_iとしたときの, 予備実験における全被験者

表 5 教示有りグループのバグ発見率

被験者	バグ発見率
被験者 a	42.3%
被験者 b	57.1%
被験者 e	71.4%
被験者 f	85.7%
被験者 g	71.4%
被験者 i	71.4%
平均	66.7%
標準偏差	14.9

表 6 教示無しグループのバグ発見率

被験者	バグ発見率
被験者 c	57.1%
被験者 d	42.3%
被験者 h	71.4%
被験者 j	57.1%
被験者 k	71.4%
平均	60.0%
標準偏差	10.4

の平均バグ発見率 sub と，本実験における全被験者の平均バグ発見率 $main$ を以下の式で求める．

$$sub = (\sum_{i=1}^{n} S_i)/n \tag{5.2.1}$$

$$main = (\sum_{i=1}^{n} M_i)/n \tag{5.2.2}$$

次に予備実験におけるバグ発見率を本実験のバグ発見率で正規化した値である NS_i と，被験者個々の本実験でのバグ発見率の差 $DIFF_i$ を求める．本実験でのバグ発見率と NS_i の差である $DIFF_i$ は以下の式で求めることができる．

$$NS_i = (main/sub) * S_i \tag{5.2.3}$$

$$DIFF_i = M_i - NS_i \tag{5.2.4}$$

表7に教示有りグループの結果を，表8に教示無しグループの結果を示す.

表から教示有りのグループは予備実験に比べて本実験でバグ発見効率が6.67%上昇しているといえる．また，教示無しのグループのバグ発見率の差は0.00%と，向上しなかった．

表 7 教示有りグループのバグ発見率の差

被験者	バグ発見率の差 (%)
被験者 a	-17.14
被験者 b	18.92
被験者 e	4.64
被験者 f	10.71
被験者 g	11.43
被験者 i	11.43
平均	6.67
標準偏差	12.8

表 8 教示無しグループのバグ発見率の差

被験者	バグ発見率の差 (%)
被験者 c	-17.86
被験者 d	-2.14
被験者 h	18.93
被験者 j	4.64
被験者 k	-3.57
平均	0.00
標準偏差	13.4

Welch の検定の結果，$p = 0.42$ で有意な差は見られなかった．しかし，グループ G2 は全く変化せず，グループ G1 のみ平均バグ発見率が 6.67%も上昇したという点から，被験者数を増やして検定をすれば有意差を見ることができると考えられる．

5.3 作業時間とバグ発見率

教示有りグループと教示無しグループの作業中の視線の動きを確認した．図 8 に教示有りグループと教示無しグループのレビュー時間とバグ発見率の遷移を示す. 図 8 において，横軸はレビュー開始からの経過時間（分），縦軸はバグ発見率を表している．

図 8 から分かる様に，教示有りグループは教示無しグループと比べレビュー開始から 10 分までのバグ発見率が低い．また，10 分から 15 分頃までは 2 つのグループの作業効率に明確な差がないことがわかる．しかし，作業開始から 15 分経過以降の発見効率は教示有りグループの方が教示無しに比べて高く，最大で 13.9%の差が見られた．2 つのグループに対して Welch の検定を行った結果，$p = 0.43$ で有意な差は見られなかった．今後，被験者数を増やした実験で統計的に有意な向上かを調査することで，より提案手法の有効性を明確にすることができると考えられる．

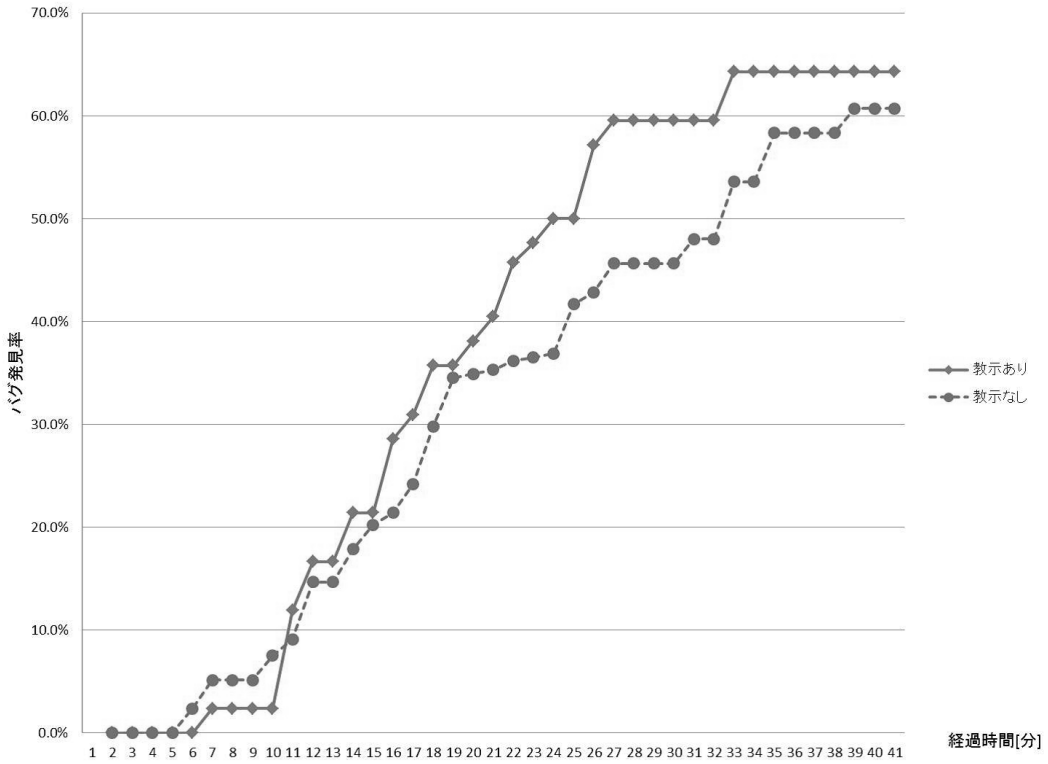

図 8 レビュー時間とバグ発見率

以上の結果から，最初の10分では，教示有りのグループはレビューの最初に設計書を丁寧に読むことを指示されているため，1つ目のバグ指摘までに時間がかかったためと予測できる．しかし，その後は教示有りグループが設計書やソースコードに目を通し終え，プログラムを理解してからレビューを開始したため，教示無しグループに比べ効率的にプログラムの理解やバグの指摘をでき，バグ発見率が急激に上昇したと考えられる．

被験者の数が少ないため，Welchの検定による有意差は見られなかったものの，レビュー時の読み方を教示することでレビュー効率を向上させることができるといえる．

次に，実際に教示有りグループが教示内容通り提示物に目を通しているかを確認するため，作業開始から10分間の被験者が注目した提示物の割合をグループ別に平均をとり，表9に示した．

本実験に使用したJavaプログラムにおいて，Main.javaは104行のソースコードであり，House.java（34行）に比べ比較的長いプログラムである．そのため，表9の結果から分かるようにMain.javaに掛ける注目時間が，一番長いことは作業者がプログラムを理解する上で妥当であるといえる．

次に，レビュー中に教示有りグループの作業者が表示した提示物の時間推移を図9に示す.

表 9 作業開始から 10 分間の被験者の注目度合い

	教示有り [%]	教示無し [%]
設計書	35.0	45.0
Main.java	50.4	39.8
House.java	11.6	19.4
bukken.dat	3.0	1.7

図 9 では縦軸が提示物を，横軸が時間推移を示しており，被験者が注目している提示物を示している．図から被験者が作業開始時に指示通り設計書を読み，提示物全体に目を通していることが確認できる．

図 10 に教示無しグループの作業者が表示した提示物の時間推移を示す.

図 10 も図 9 と同様に被験者が注目している提示物を示している．図から分かるように，教示有りグループに比べ，教示無しグループの実験中の画面切り替えはせわしなく，短い時間中に何度も見直していることがわかる．

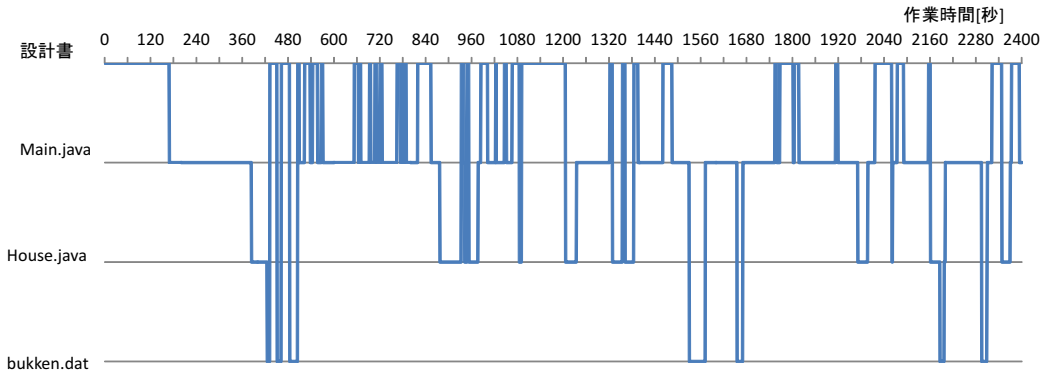

図 9 教示有りの視線の動き

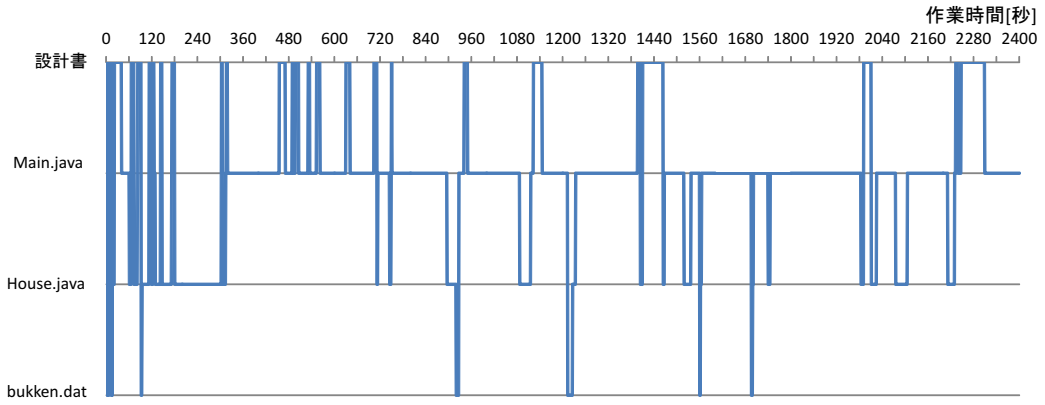

図 10 教示無しの視線の動き

6 おわりに

本研究ではソースコードレビューにおいて作業者にレビュー開始時の読み方を教示することで作業効率が上昇するかを検証した．実験では先行研究で開発された視線計測環境を用い，視線の動かし方について教示をした被験者と教示をしなかった被験者で作業効率に差は見られたかを検証した．結果として読み方の教示をした被験者の方が発見したバグ数が教示しなかった被験者に比べ高くなることが分かった．また，読み方の教示をした被験者は教示前に比べ，バグ発見率が上昇したが，教示しなかった被験者はバグ発見率が変わらなかったというデータも得られた．これはレビュー開始時に設計書を精読した後，プログラム全体に目を通したためにプログラムの内容理解が容易になったためと考えられる．さらに，視線の動かし方を教示したグループは教示した手順を遂行した直後からのバグ発見率が高いということも分かった．これは被験者がプログラムの内容を理解し，バグを発見することが容易になったためと考えられる．

これらのことからレビュー開始時の読み方の教示は，プログラムの内容理解を助けるためのツールになりうると考えられる．今後の課題は実験の際の被験者数を増やし，多くのデータを取ることで今回の結果に有意差があるかを明らかにすることである．

謝辞

本研究を進めるに当たり，多くの方々にご指導，ご協力を賜りました．特に，奈良工業高等専門学校情報工学科上野秀剛助教には指導教員として常日頃よりご指導やご助言を頂きました．また，日常生活においてもご配慮，ご助言いただきましたことを心より感謝申し上げます.

また，奈良工業高等専門学校情報工学科内田眞司准教授には貴重ご意見やご指摘を頂き，心から感謝申し上げます.

本研究を進めるに当たり，実験の被験者としてご協力いただいた方々に心より感謝申し上げます.

参考文献

[1] M. E. Fagan : “Design and Code Inspection to Reduce Errors in Program Development，” IBM Systems Journal，Vol.15，No.3，pp. 182-211(1976).

[2] F. J. Shull : “Developing Techniques for Using Software Documents: A Series of Empirical Studies，” PhD thesis,Univ. of Maryland(1988).

[3] T. Thelin, P. Runeson, and B. Regnell : “Usage-based reading — An experiment to guide reviewers with use cases,” Information and Software Technology, Vol. 43, No. 15, pp. 925-938(2001).

[4] 松川文一，Giedre Sabaliauskaite，楠本真二，井上克郎 :“UMLで記述された設計仕様書を対象としたレビュー手法CBRとPBRの比較評価実験”，オブジェクト指向最前線2002，pp. 67-74 (2002).

[5] A. A. Porter，L. Votta:“Comparing Detection Methods for Software Requirements Inspection: A replication using professional subjects，” Empirical Software Engineering: An International Journal，Vol.3，No.4， pp. 355-380(1988).

[6] F. Lanubile, G. Visaggio : “Evaluating Defect Detection Techniques for Software Requirements Inspections,” ISERN Technical Report-00-08(2000).

[7] T. Thelin, P. Runeson, and C. Wohlin : “An Experimental Comparison of Usage-Based and Checklist- Based Reading,” IEEE Transaction on Software Engineering, Vol. 29, No. 8, pp. 687-704(2003).

[8] 村田厚生，森若誠 :“危険予知課題における運転者の視覚情報処理特性 - 運転初心者と運転熟練者の比較”，人間工学，Vol.46，No.6，pp. 393-397(2010).

[9] 岩城朝厚，前野浩孝，六十谷伸樹，中田早苗,曽我真人，松田憲幸，高木佐恵子，瀧寛和，吉本富士市 :“学習者のデッサン描画時における腕動作・視線・認識の分析”，第19回人工知能学会全国大会，Vol.19(2005).

[10] S. Zhai，C. Morimoto，and S. Ihde : “Manual and gaze input cascaded (MAGIC) pointing，” In Proceedings of The SIGCHI Conference on Human Factors In Computing Systems ‘ 99，pp. 246-253(1999).

[11] 中道上，島和之，中村匡秀，松本健一 :“視線を利用したWebユーザビリティ評価環境”，情報処理学会論文誌，Vol.44，No.11，pp. 2575-2586 (2003).

[12] Randy Stein，Susan E. Brennan : “Another Person’s Gaze as a Cue in Solving Programming Problems，”In Proceedings of The 6th International Conference on Multimodal Interface，pp. 9-15(2004).

[13] 上野秀剛 :“プログラマの視線を用いたコードレビュー性能の要因分析”, 奈良先端科学技術大学院大学修士論文 (2006).

[14] 吉本温 :“コードレビューにおける読み方とレビュー効率”, 奈良工業高等専門学校情報工学科 2012 年度卒業論文 (2013).

[15] 木浦幹雄 , 大平雅雄 , 上野秀剛 , 松本健一 :“Webjig:ユーザ行動とユーザ画面の関連付けによる動的 Web サイト利用者の行動可視化システムの開発および評価 , ”情報処理学会論文誌 , Vol.51,No.1,pp. 204-205(2010).

[16] T.Thelin, P.Runeson, C.Wohlin : “Prioritized Use Cases as A Vehicle for Software Inspections,”IEEE Software, Vol.20, No.4, pp. 30-33(2003).

付録

予備実験で使用したソースコード

練習用プログラム

Main.java

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args){

        Scanner stdIn = new Scanner(System.in);

        System.out.println("1からnまでの和を求めます。 ");

        int n;
        System.out.print("nの値 : ");
        m = stdIn.nextInt();

        System.out.println("1から" + n + "までの和は" + Sum.sumUp(n) + "です。");
    }

}
```

Sum.java

```
public class Sum {
    public static int sumUp(int n){
        int sum = 1;
        int i = 1;

        while (i <= n){
            sum += i;
            i--;
        }
        return sum;

    }

}
```

設計書

1からnまでの整数の和を求めて表示するプログラム.

Mainクラスには以下のメソッドがある.

public static void main(String[] args)
nを読み取り, 1からnまでの整数の和を出力する.

Sumクラスには以下のメソッドがある.

public static int sum(int n)
1 から n までの整数の和を求める.

誤り

Sum.java より

- メソッド Sum において変数 sum は 0 で初期化するべきなのに 1 で初期化されている.

- メソッド Sum において while 文内の変数 i が加算されるべきなのに減算されている.

予備実験プログラム No.1

Account.java

```
public class Account {
    public String name;
    public String no;
    public double  balance;
    public double interest;

    public Account(String name, String no, int balance, double interest) {
        this.name = name;
        this.no = no;
        this.balance = balance;
        this.interest = interest;
    }

    public boolean isSufficientAmount(int amount) {
        return balance >= amount;
    }

    public void deposit(int amount) {
        balance += amount;
    }

    public boolean withdraw(int amount) {
        if (!isSufficientAmount(amount)) {
            return true;
        }
        balance -= amount;
        return false ;
    }

    public void calcInterest() {
        balance += balance * interest;
    }
```

```
    public String toString() {
        return name + "(" + no + ") 残高:" + balance + ", 金利:" + interest * 100 + "%";
    }
}
```

Bank.java

```
public class Bank {

    public static void main(String[] args) {
        Account list[] = new Account[3];
        list[0] = new Account("Ichiro", "0001", 0, 0.02);
        list[1] = new Account("Jiro", "0002", 10000, 0.025);
        list[2] = new Account("Saburo", "0003", 1000, 0.025);

        showAccount(list);

        System.out.println("\n" + list.length + "件の口座に10000円追加します. ");
        for (int i = 0; i >  list.length; i-- ) {
            list[i].deposit(10000);
        }

        System.out.println("追加後");
        showAccount(list);

        System.out.println("\n" + list.length + "件の金利の計算をします. ");
        for (int i = 0; i < list.length; i++) {
            list[i].calcInterest();
        }

        System.out.println("\n金利計算後");
        showAccount(list);

        System.out.println("\n引き落とし可能な口座から20000円引き落とします. ");
        for (int i = 0; i < list.length; i++) {
            list[i].withdraw(20000);
        }

        System.out.println("\n引き落とし後");
        showAccount(list);
    }

    private static void showAccount(Account list[]) {
        for (int i = 0; i < list.length; i++) {
            System.out.println(list[i].toString());
        }
    }
}
```

設計書

Accountクラスと Bankクラスから構成されるプログラムは貯金の残高を管理する. Accountクラスには以下のフィールド, メソッドがある.

public String name 口座所有者の氏名

public String no 口座番号
public int balance 口座残高
public double interest 金利

public Account(String name , String no, int balance, double interest)
口座情報をフィールドに格納するコンストラクタ

public Boolean isSufficientAmount(int amount)
口座の残高が amount 以上あるか確認する.
amount 以上あれば true, なければ false を返す.

public Boolean withdraw(int amount)
残高が十分にあるか isSufficientAmount メソッドを用いて確認し,
十分にあれば口座の残高から amount を減らし true を返す.
残高が amount より少なければ何もせず false を返す.

public void calcInterest()
interest を元に利子を計算し balance に加える. 利子の小数点以下は切り捨てる.
Math クラスを利用する.

public String toString()
口座所有者の氏名と口座番号, 残高, 金利を返す.

Bank クラスには以下のメソッドがある.

public static void main(String[] args)
最初にすべての口座を表示し, 全ての口座に 10000 円追加し, 追加後の残高を表示する.
金利の計算をし, 計算結果を表示する.
また, 引き落とし可能な口座から 20000 円引き落とし, 引き落とし後の残高を表示する.

private static void showAccount(Account list[])
口座のリストを表示する

誤り

Account.java より

- 口座残高がint型でなくdouble型で定義されている．

- メソッドwithdrawにおいて残高が十分にあれば口座の残高からamoutを減らし，trueを返し，残高がamountより少なければ何もせずfalseを返す条件が逆になっている．

- メソッドcalcInterestにおいてMathクラスを使用していない．また，利子の小数点以下が切り捨てられていない．
  Bank.javaより

- メソッドshowAccountよりfor文内の条件式の不等号の向きが逆になっている

予備実験プログラムNo.2

July.java

```
public class July {

    public static void main(String[] args) {
        Day[] july = new Day[30];

        for (int i = 0; i < july.length; i++) {
            july[i] = new Day(2012 , 7, i + 1);
        }

        System.out.println("7月の全要素：");
        for (int i = 0; i < july.length; i++) {
            System.out.println(july[i].toString());
        }
    }

}
```

Day.java

```
public class Day {
    private int year;
    private int month;
    private int date;


    public Day(int year, int month, int date) {
        this.year = year;
        this.month = month;
```

```
    }


    int dayOfWeek() {
        int y = year;
        int m = month;
        if (m == 1 || m == 2) {
            y--;
            m - = 12;
        }
        return (y + y / 4 - y / 100 + y / 400 + (13 * m + 8) / 5 + date) \% 7;
     }


    @Override
    public String toString() {
        String[] wd = { "月", "火", "水", "木", "金", "土" ,"日" };
        return String.format("\%d 年 \%d 月 \%2d 日 (\%s)",year, month, date, wd[dayOfWeek()]);
    }
}
```

設計書

2013年の7月1日から31日までの年月日と曜日を1行に1日ずつ表示するプログラム.

Dayクラスには以下のフィールドとメソッドがある.

private int year 年

private int month　月

private int date　日

public Day(int year, int month, int date)

受け取った年月日をフィールドに格納する.

int dayOfWeek()

ツェラーの公式を用いて年月日から曜日を計算する. ツェラーの公式は月(m)が1または2の場合, 年(y)から1引き, 月に12たしたうえで以下の公式を使う.

(y + y / 4 - y / 100 + y / 400 + (13 * m + 8) / 5 + date) % 7;

public String toString()

年月日と曜日の文字列を返す.

Julyクラスには以下のメソッドがある.

public static void main(String[] args)

2013年7月を表すDayクラスの配列を用意し，年月日と曜日を順に表示する．

誤り

July.javaより

- 配列Dayの配列数が足りない．

- 2013年7月を表示するプログラムであるのに2012年と表示される．

Day.javaより

- メソッドDayにおいてdateが定義されていない．

- メソッドDayOfWeek内のif文においてmは加算されるべきなのに減算されている．

- メソッドtoStringにおいて曜日の表示が月曜日始まりになっている．

本実験で使用したソースコード

練習用プログラム

Main.java

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args) {
        Scanner stdIn = new Scanner(System.in);
        System.out.println("n段のピラミッドを表示します");
        System.out.print("段数を入力してください：");
        int n;
        n = stdIn.nextInt();
     Pyramid(n);
    }
}
```

Pyramid.java

```
public class Pyramid{
    public static int pyramid(int n){
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j < i + n ; j++) {
```

```
                if (j <= n - i) {
                    System.out.print("*");
                } else {
                System.out.print(" ");
                }
            }
        System.out.println();
        }
    }
}
```

設計書

n段のピラミッドを作成するプログラム.

Mainクラスには以下のメソッドがある.

public static void main(String[] args)
何段のピラミッドを作成するか読み込み, ピラミッドを出力する.
3段のピラミッドの例を以下に示す

```
**
****
```

Pyramidクラスには以下のメソッドがある.

public static int pyramid(int n)
ピラミッドの段数を読み込み, 表示する.

誤り

Pyramid.javaより

- *と空白の表示が逆.

本実験プログラム

House.java

```
public class House {
    private int area;
    private double distance;
    private int rent;
    private String name;
```

```
    public House(String n, int a, double d, int r) {
        area = a;
        distance = d;
        rent = r;
        name = n;

    }

    public double getDistance() {
        return distance;
    }

    public int getArea() {
        return area;
    }

    public int getRent() {
        return rent;
    }

    public void showHouse() {
        System.out.println(name + "\t" + area + "\t" + distance);

    }

}
```

Main.java

```
import java.io.*;
import java.util.Scanner;

public class Main
 {

    private int MAXHOUSE = 255;

    public static void main(String[] args) {

        String fileName = "bukken.dat";
        int i;
        House[] data = null;
        try {
            data = readBukkenFile(fileName);
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("ファイルが開けません. ");
            System.exit(1);
        }

        if (data.length > 0) {
            Scanner stdIn = new Scanner(System.in);
            while (true) {
                    System.out.println("#条件入力【坪数[0], 駅からの距離(km) [1], 家賃
[2], 終了[9]】: ");
                    int selectNum = stdIn.nextInt();
                    switch (selectNum) {
```

```
                        case 0:
                            System.out.println("坪数で検索します. ");
                            break;
                        case 1:
                            System.out.println("距離で検索します. ");
                            break;
                        case 2:
                            System.out.println("家賃で検索します. ");
                            break;
                        case 9:
                            System.out.println("終了します. ");
                            stdIn.close();
                            System.exit(1);
                        default:
                            System.out.println("入力が間違っています. ");
                            continue;
                    }

                    System.out.println("#検索範囲入力【上限】: ");
                    int lower = stdIn.nextInt();
                    System.out.println("#検索範囲入力【下限】: ");
                    int upper = stdIn.nextInt();
                    bukkenSearch(data, selectNum, lower, upper);
            }

        } else {
            System.out.println("データファイルは空です. ");
        }
    }

    static void bukkenSearch(House blist[], int selectNum, int lower, int upper) {
        System.out.println("物件名 坪数  距離  家賃");

        for (int i = 0; i < MAXHOUSE && blist[i] != null; i++) {
            switch (selectNum) {
                case 0:
                    if (blist[i].getArea() >= lower ||  blist[i].getArea() <= upper) {
                        blist[i].showHouse();
                    }
                    break;
                case 1:
                    if (blist[i].getDistance() >= (double) lower &&
blist[i].getDistance() <= (double) upper) {
                        blist[i].showHouse();
                    }
                    break;
                case 2:
                    if (blist[i].getRent() >= lower && blist[i].getRent() <= upper) {
                        blist[i].showHouse();
                    }
                    break;
                }

        }
    }

    static House[] readBukkenFile(String filename) throws IOException {
        House blist[] = new House[MAXHOUSE];
        FileReader fr = new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        String line = null;
```

```
        int i = 0;

        while ((line = br.readLine()) != null)  {
            String[] words = line.split(",");
            String name = words[0];
            int area = Integer.valueOf(words[1]);
            double distance = Double.valueOf(words[2]);
            int rent = Integer.valueOf(words[3]);
            blist[i] = new House(name, area, distance, rent);
            i++;
        }

     br.close();
        fr.close();
        return blist;
    }
}
```

bukken.dat

```
LionHouse,40,5.7,150000
PensionTamada,10,2.5,70000
DormitoryNaka,25,3.1,80000
PalaceIgaki,50,1.1,350000
RabbitHome,7,1.7,40000
YamadaSo,12,3.2,60000
OtogiriSo,45,10.0,120000
PrismTanaka,31,15.1,115000
FrontIshii,25,2.2,100000
TowerSuit,33,0.7,250000
HillSasaki,18,7.0,80000
MountTakata,15,3.2,55000
HeavenCry,17,5.5,100000
TearHamada,62,4.8,330000
SkyCourtArima,35,3.2,150000
VogueYoshino,6 2.7,52000
GrandYukari,8,4.1,35000
BeautyUmezawa,21,7.2,72000
SkynetTunoda,9,2.8,68000
RandomMurakami,11,0.9,170000
```

設計書

ファイルbukken.datから物件データを得る.
Main.javaとHouse.javaを用いて物件データから指定した条件に一致する物件を表示する.

Mainクラスには以下のメソッドがある.

public static void main(String[] args)
readBukkenFileメソッドを用いてファイルから物件データ(data)を読み込む.
他のメソッドはdataから物件データを得る.
ファイルが開けなかった場合は「ファイルが開けません.」と返し, 実行を終了する.
ファイルが空だった場合, その旨を表示して終了する.

ユーザ入力(検索条件，検索範囲)をbukkenSearchメソッドに渡し，指定された処理を行う.
また，入力が正しくなかった場合,「入力が間違っています.」と返し，実行を続ける.
ユーザが終了[9]を選択するまで検索を繰り返す.

static House[] readBukkenFile(String filename)
物件ファイル(filename)から物件データを読み込み，物件リスト(blist)に
セットする．読み込む最大データ数は100件とする．ファイルに100件以上のデータが
含まれている場合は100件以降のデータは読み込まない．
個々の物件データはwhile文を用いてblistにセットする．
返り値は，blist.

static void bukkenSearch(House blist[], int selectNum, int lower, int upper)
検索条件(selectNum)および検索範囲(lower, upper)を元に，受け取った物件リスト(blist)を検索し，
条件にマッチする物件を表示する．個々の物件データはshowHouseメソッドで表示する．
selectNum:坪数[0]，駅からの距離(km)[1]，家賃[2]
lower:範囲下限，upper:範囲上限

Houseクラスには以下のメソッドがある．

public double getDistance()
駅からの距離を返す.

public int getArea()
坪数を返す.

public int getRent()
家賃を返す.

public void showHouse()
物件の情報を名前，坪数，駅からの距離，家賃の順に一行で表示する．

誤り

House.java より

- rent の出力が足りない

Main.java より

- main メソッドにおいて不要な変数 i が宣言されている.

- main メソッドにおいて検索範囲の上限と下限が逆になっている.

- メソッド bukkenSearch において switch 文内の case 0 における if 文で積が和になっている.

- メソッド readBukkenFile において while 文の中身 i ¡ MAXHOUSE が抜けている.

実験時に作業者に提示したコードチェックリストを以下に示す.

```
\textbf{コードチェックリスト}

完全性
  設計された内容は完全にコーディングされているか


初期化
  プログラムの以下の時点で変数とパラメータが初期化されているか
    プログラム起動時
    ループ開始時
    クラス呼び出し時


メソッド呼び出し
  パラメータの型, 順序は正しいか
  適切な戻り値が返されているか


演算
  =, ==, || などの演算子は正しく使われているか
  不等号の向きや統合の有無は正しいか
  四則演算の優先順位や () は正しく使用されているか


データ・ファイル
  データの型, 値は正しく設定されているか
  データへのアクセスは正しく行われているか
```

すべてのファイルは以下の条件を満たしているか
- 正しく宣言されている
- オープンされている
- クローズされている